A VISUALIZATION FRAMEWORK FOR PATIENT DATA AND ITS ENVIRONMENT

by

PAVANI AYYAGARI

B. Tech., Jawaharlal Nehru Technological University, India, 2007

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2010

Approved by:

Major Professor
Dr. Gurdip Singh

Copyright

Abstract

A Health Care System is targeted to provide well-monitored patient care. ‘A Visualization Framework for Patient data and its environment’ offers a broad portfolio of patient monitoring to help and improve patient care.

‘A Visualization Framework for Patient data and its environment’ is an application to monitor and analyze any patient’s activity visually through a period of time in conjunction with the surroundings aspects like the room temperature, status of lights in the room, etc. in a healthcare system. A set of sensors equipped with each patient record data, pertaining to the patient’s movement and location in addition to a few other sensor values like temperature and light sensors for recording the room temperature, status of lights in the room respectively. On accepting the activity time bounds as the input, the application retrieves the appropriate values from the database and displays the patient position as a continuous stream of images in association with a slider along with the temperature values. A floor map of the hospital, similar to a blueprint model, is portrayed along with graphical display of lights, in conjunction with the slider. The patient locations are depicted on the map by minute icons with a patient id associated with the each of the icons for identification purposes. Individual window frames for each patient, displaying a patient’s position, enable the user to customize monitoring to specific patients at any instance of time and thus keep track of every move of the patient over a considerate period of time. The locations of the patients on the map, the lights in the rooms depicted on the floor map, the patient position in the individual windows and the temperature are all in synchronization with the slider whose movement is a function of time. The application allows monitoring of values that correspond closely to real-time data values thus maximizing the scope of improvements in the patient’s progress.

The application is implemented on a Java Platform using Swings and is expected to handle considerable amounts of data up to two days.

Table of Contents

List of Figures

Acknowledgements

I would like to thank my major professor Dr. Gurdip Singh for his constant help, kind patience, academic advice and guidance throughout the project and my entire Masters program.

I also thank Dr. Daniel Andresen and Dr. Torben Amtoft for serving in my committee and for their support and timely cooperation.

Finally, I would like to thank my family, Dinesh Challa, Samuel Kummary, my supervisor Terry Stout, my friends Sumanthi Chakrapu, Chalapathi Pemmaraju, Meenakshi Sharma, Rohit Parimi and Sandeep Pulluri for their unflinching support and cooperation.

Dedication

I dedicate this work to my parents for their constant support and encouragement throughout my entire career and whose cooperation played a major role in the success of the project.

CHAPTER 1 - Introduction

1.1 Overview

An application that integrates the monitoring of the patient as well as the surrounding metrics like temperature, pressure, status of lights in the room etc. is an efficient contribution to a Health Care System's infrastructure. This application thus provides an effective solution to analyze the activities of the patients in a Health Care System. The application is implemented in Swing and provides a visualization framework similar to a miniature version of a multimedia player supporting video format. The project aims at displaying the patient positions as a continuous stream of images in addition to displaying the movement of the patient on a blueprint model of the floor plan along with all the other sensor values like the temperature of the room, status of lights in the room etc., thus maintaining a synchronization of all the displayed values. However, the display is not just confined to a single patient. Multiple patients can be represented on the floor map of the Health Care System with their respective sensor values along with a provision to select the respective individual's position in a separate window. The application is expected to handle any number of values up to two days which is implemented by parsing the given time into regular intervals and retrieving the values from the database in those short intervals. A functionality of pausing and continuing the movement of slider which is associated with all the values being displayed, in addition to the feature of dragging the slider to any desirable position forward or backward with respect to time has also been implemented.

1.2 Motivation

A Health Care System's major concern related to a patient is the patient's progress. For a Health Care System to grow successfully, a patient should be treated with good care by diagnosing the patient's progress regularly and constantly monitoring the patient's activities. A patient's physical condition can trigger the need for a complete dedication on a person's behalf in order to remain in reach of the patient when needed. A critical situation can also demand a constant supervision of the patient. However, allocating one person to every few patients involves abundant manpower and resources on the Health Care's behalf which is not cost-effective. Hence, the need arises to build an application in order to flexibly monitor the patients thus reducing on manpower while efficiently tracking the person from a convenient distance.

More so, it can also be the cause for avoidance of an emergency situation that might lead to a crisis as a reason of delayed response on behalf of the Health Care System. A patient can be associated with a set of sensors preferably sunspots that can be configured to detect the patient's every possible position. Projection of these images as a display, attributes to real time streaming of the patient's progress. However, such an application does not allow monitoring the patient at a later stage or analysis of the patient's progress over a period of time. Hence the need to create an application that can retrieve these values from the database at any time and display them. Eventually it also paves way for an operational situation where a patient can never be left unattended.

1.3 Objectives

The application's main objective is to deliver a product that possesses a mechanism to project every position of the patient as a continuous stream of images over a desired period of time. This is done by retrieving values from the database and projecting them on to a graphical interface. Thus the application is anticipated to accomplish the following objectives:

- The patient's positions as well as the other sensors associated with the patient's surroundings are displayed as a function of time.
- The patient is depicted graphically on a blueprint model of the Health Care System [16] so as to track the patient's location at every instant of time.
- The other sensor values like room temperature, status of lights in the rooms, etc are also portrayed on the map that corresponds to a blueprint model of the floor plan of the Health Care System.
- A conventional procedure of opting to monitor the positions of only selective patients is attributed with the application in order to monitor the patients separately and selectively in separate windows.
- In addition to projection of the activity as a function of time, the application is also incorporated with the feature of toggling the streaming between the given time period.
- The input accepted from the user is parsed, after validation of the data, into regular intervals in order to avoid any overhead during the retrieval of the values from the

database, thus allowing the application to handle large values of data without any room for latency.

- The application responds to any changes made to the database during the execution of the project by updating the display appropriately.

CHAPTER 2 - Planning & Requirements

As the next step in the Software Development life cycle, the basic plan is designed and the requirements are analyzed in order to have a crystallized view of the problem description. This chapter commences with the identification of the problem description, potential problems that can possibly occur which lead to some assumptions and proceeds by listing out the technical requirements needed for the application including the various technologies used and proposes an initial approach.

2.1 Problem Description

Although a patient is equipped with sensors that record the necessary sensor values for location, temperature and light values, the values returned by the sunspots that are used to detect the changes can only be monitored as real-time entities. In other words, the sensors that detect the values, display them instantaneously with no time intervention. This renders no room for monitoring the patient's activity or viewing it at convenience at a later time. Also, designing an application with all the above mentioned values, while simultaneously dealing with real-time data accounts to a very complex application ultimately producing certain undesired results. Hence the problem is to be handled by proposing to store the values recorded by the sensors, in a database so as to be able to access and display through a Graphical User Interface (GUI) in a conventional and a convenient manner.

2.2 Assumptions

Though the application is configured to run under any circumstances, it assumes a certain conditional cases. As the application's major input is the database, most of the assumptions pertain to it. The following account to the assumptions made by the project.

- The various postures of the patient, retrieved from the database are familiar to the application thus having an associated image file for each different posture.
- The application can handle display of information at an interval different from that being recorded in the database. In other words, the patient positions, location co-ordinates, room temperature, status of lights can all be recorded at different intervals. However, the interval for each entity can is to be recorded at regular intervals only.

- The floor map or a blueprint model of the hospital is already known to the application.
- The database is expected to hold legitimate and appropriate values especially those pertaining to position coordinates.

2.3 Requirement Specifications

The following section identifies the required hardware specifications and the software tools required for the application to be implemented.

2.3.1 Hardware Specifications

The application was executed in the following environment. The application is expected to run in an environment with a configuration lesser than the below mentioned one. However, to meet the results and performance displayed in the following chapters, a minimum of the following configurations would be required.

Processor: Intel Core 2 Duo

RAM: 2GB

Memory: 2 MB

Processor Speed: 2 GHz

2.3.2 Software Specifications

The following section lists the software tools that have been used to develop the application.

Operating System: Windows XP

Platform: J2SE 5.0

Language: Java (JDK 1.6)

IDE: NetBeans 6.0

Database: MySQL 5.1

Apart from the above mentioned operating system, Windows Vista, Windows 7 and Linux can also be used. NetBeans accounts to a very comfortable IDE (Integrated Development Environment) when using Java Swing. Hence, the purpose of its use. Any IDE that provides the feature of implementing Swing can be used. The use of an IDE is also optional. The application

is compatible with any other database like Oracle, MS access, SQL server too which leads to a much wider flexibility in choosing the database.

2.4 Technologies

A set of tools and technologies have been used to deploy the application. The following could be listed as some of the major technologies used as a process of implementation.

J2SE: Java SE is a widely used platform for programming in the Java language. It is the Java Platform used to deploy portable applications for general use. In practical terms, Java SE consists of a virtual machine, which must be used to run Java programs, together with a set of libraries (or packages) needed to allow the use of file systems, networks, graphical interfaces, and so on, from within those programs. [1]

Swing: It is an extension library to the AWT which includes new and improved components that enhance the look and functionality of GUIs. Swing can be used to build Standalone swing GUI Apps as well as Servlets and Applets. It employs a model-view-controller design architecture. Swing is more portable and more flexible than AWT. [2]

Jslider: It is a component that lets the user graphically select a value by sliding a knob within a bounded interval. The slider can show both major tick marks and minor tick marks between them. The number of pixels between the tick marks can also be controlled. [3]

JavaFX: JavaFX is an expressive client platform that provides a unified development and deployment model for building rich client applications. It combines the best capabilities of the Java platform with comprehensive, immersive media functionality into an intuitive and comprehensive, one-stop development environment. It enables easy integration of audio and video, graphics, rich text, and Web services for creating and delivering rich Internet experiences across all screens of life. [4]

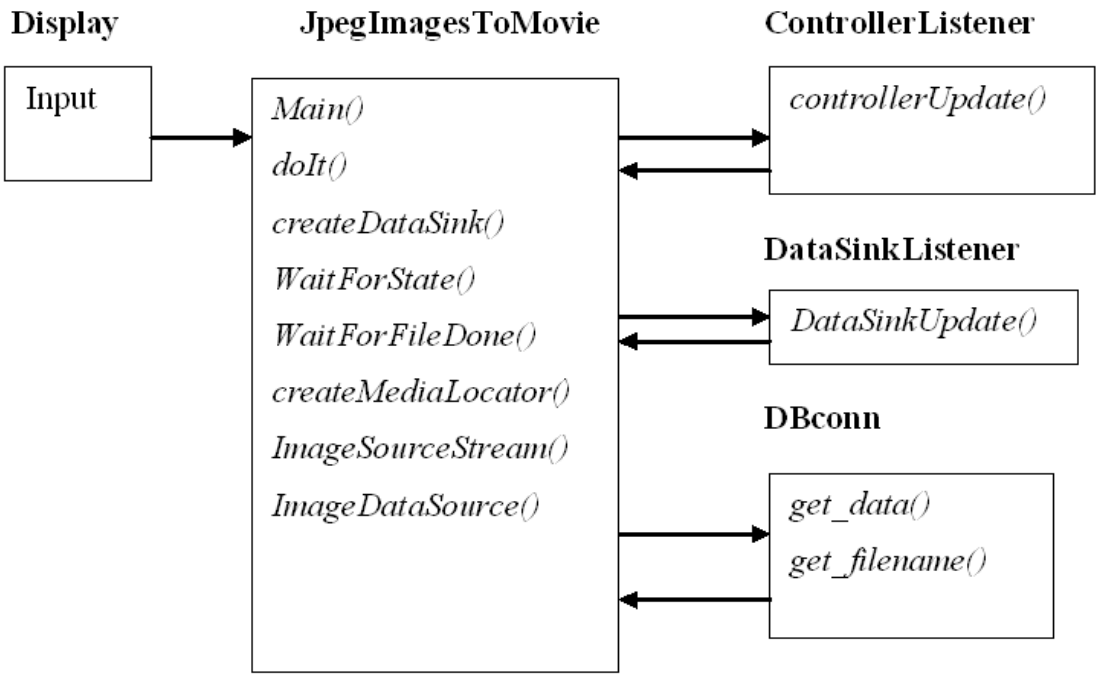
JDBC: The JDBC API provides programmatic access to relational data from the Java programming language. Using the JDBC API, applications can execute SQL statements, retrieve results, and propagate changes back to an underlying data source. It is part of the Java platform J2SE. It establishes a bridge between the Java application and the database. [5]

MySQL: MySQL is a relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. [6]

2.5 Initial Approach

An initial approach was made to accomplish the desired results by implementing the application on JavaFX Platform with JMF (Java Media Framework) Technology. The application's behavior identifies with the current implemented procedure in the way that the time period accepted from the user was parsed into shorter intervals to retrieve the values from the database. This approach creates a processor which, with the help of inner classes and methods, creates a *.mov* file which is the output file that can be found in the destination directory. This method makes use of DataStreams that read from a list of JPEG image files and turn them into a stream of JMF buffers. The processor is configured to aggregate all these buffers in a synchronized manner in order to generate an executable file that can be played as a continuous stream through a multimedia player supporting a video format. However, this approach was confined to very small vales of data pertaining to only a few hours as a multimedia player can only be configured to handle a video file of length up to four hours approximately. Hence this approach was considered deficient and thus the current procedure was implemented. The architecture of this approach can be best explained by the following diagram.

Figure 2.1 Design Diagram for an initial approach



CHAPTER 3 - Design

The application consists of multiple java files, classes and interfaces and many references between them. Thus, this chapter deals with the design of the application in order to put forth the underlying architecture in a structured and a presentable manner.

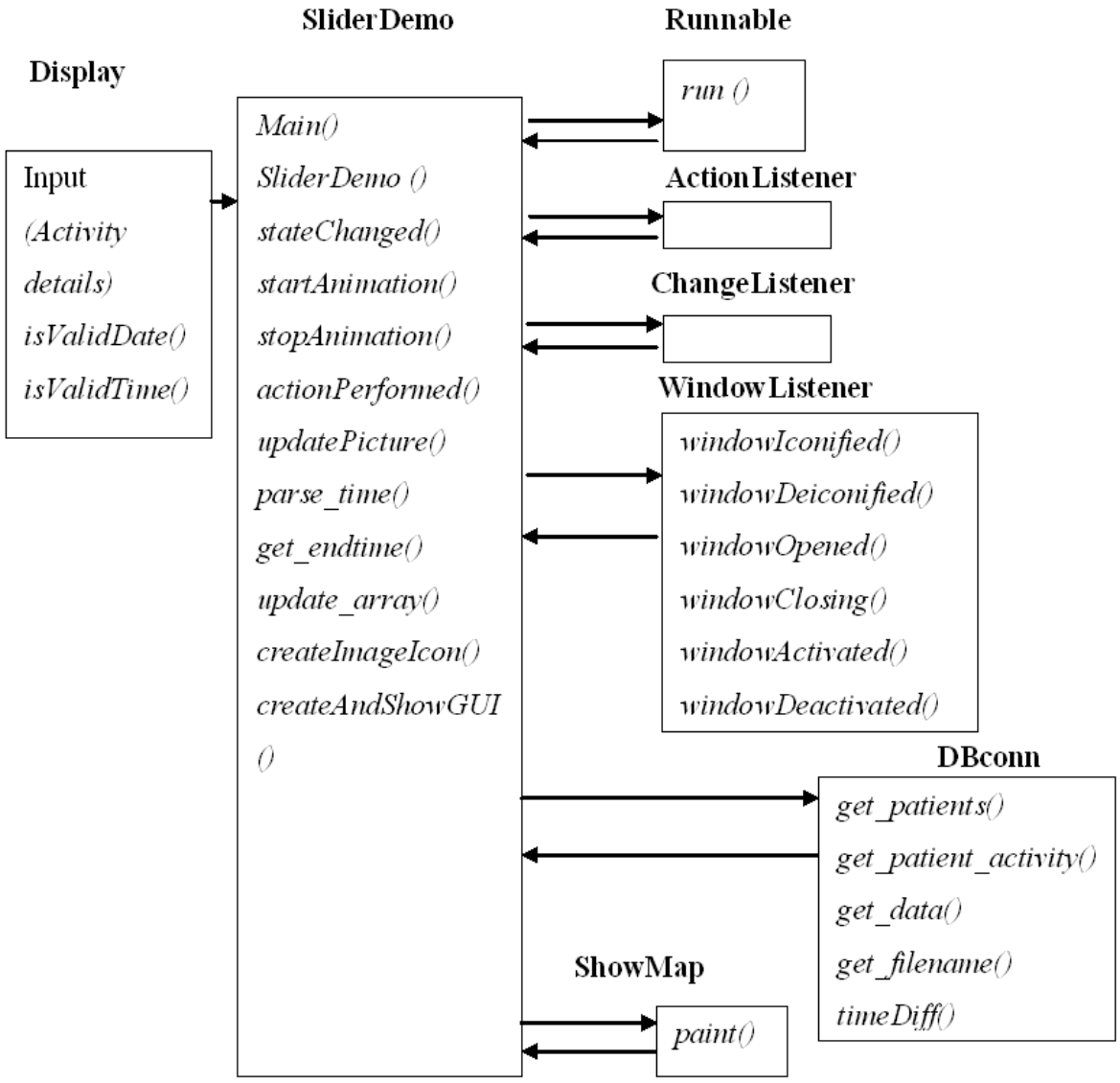
3.1 System Architecture

The underlying architecture on which the system is built is the MVC (Model-View-Controller) architecture. The MVC architecture suggests for a visual application to be divided into three separate parts for ease of use:

- A *model* that represents the data for the application.
- The *view* that accounts for the data to be represented visually.
- A *controller* that accepts the user input on the view and processes it to incorporate changes in the model.

This can be implemented with Swing as the Swing library makes immense use of the software design pattern Model/View/Controller, which separates the data being viewed from the user interface through which it is viewed. In the following application the database can be a basic representation of the model. ShowMap which represents the map and the graphical display can represent the view and the SliderDemo and Display can represent the controller that accepts input and processes it. The Display component accepts the input and passes it to SliderDemo. This accounts for the user input. SliderDemo then makes a call to Dbconn in order to receive the input to the application. It displays the output in conjunction with ShowMap. The architecture when represented systematically in the form of a diagram gives a broad overview of the system's functionalities. Thus, this can be represented as follows:

Figure 3.1 Design Diagram



The design is majorly divided into four components representing the java files with their interfaces. These could be discussed in detail as follows:

- *Display:* This component passes the input to SliderDemo. In the case of the application "A visualization framework for patient data and its environment", the input given is as follows:
  - Start Date (Date from which the activity is to be time lined)

- End Date (Date to which the activity is to be time lined) Start Time (Time from which the activity is to be time lined/monitored)
- End Time (Time to which the time lining takes place on the given date)

The application handles two different kinds of inputs. One includes the database which is assumed to have legitimate values and the required fields and the other is the user input taken from the user in order to handle limits on the time frame in which the activity is to be monitored. This class accepts the user inputs and validates them before sending them to access data from the database.

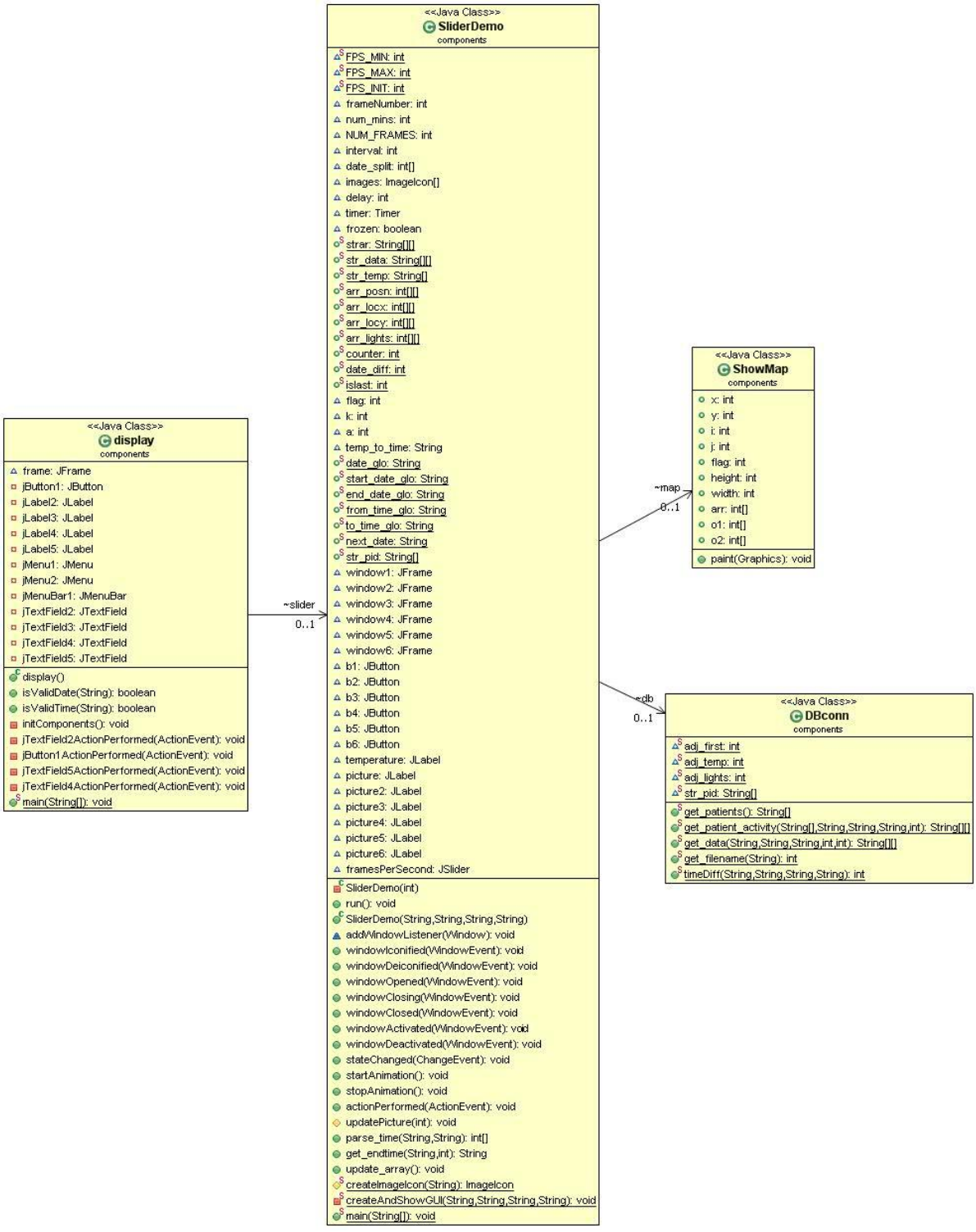
- *SliderDemo:* This component takes the input from 'display' in order to access data from the database and display it. The input values are parsed and are divided into intervals so as to decrease the overhead on the database by calling DBconn with the truncated intervals. Once these values are retrieved the ShowMap class is called in order to display the image with the updated information. It also handles display of the patient positions by processing the image files and thus, creating a continuous display of data. However it provides functionality with buttons on the display window corresponding to the patients represented on the map, which when clicked, display the position of the patient from that instance of time.
    - *Input: start date, end date, start time, end time.*
- *Dbconn:* This component is called exclusively to access values from the database. It retrieves the values by iterating in a loop in order to decrease the response time in accessing those values. It has methods that get the patients in the database and that return all the values to be displayed for the given time interval. These methods are sent to SliderDemo so as to be processed and displayed in the appropriate manner.
- *ShowMap:* This component is called by SliderDemo in order to project an image of the blueprint model of the hospital and a graphical display of the locations of all the patients on it with appropriate identifications of the patients in addition to the lights in the rooms.

3.2 Class Diagrams

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the

classes. The different components comprising of the different classes can be projected in a class diagram as follows:

Figure 3.2 Class Diagram



Display – The input accepted needs to be validated before it is processed. Hence the need for the following functions in the display class.

- *isValidDate():* This function creates an object for the SimpleDateFormat class to check for the validity of the date provided as input.
- *isValidTime():* Similar to the above function, this function creates an object for the SimpleDateFormat class to check for the validity of the time provided as input.

SliderDemo – This handles the major processing of the data provided as input and the data accessed from the database. The different methods in the SliderDemo class can be elaborated as follows:

- *update_array():* This method calculates the start time and the end time of the activity after calling a function to divide the time into intervals as the data is to be retrieved in intervals. Once the data is retrieved from the database, it is stored in an array which is updated every time this method is called by the thread which runs in the background while the animation is in progress. It also updates the start times required for retrieving the values from the database for the next time.
- *get_endtime():* This method is used to calculate the end time based on the desired interval, when given the start time. It adds the given interval to the given start time and returns the end time. If the time exceeds the day, the end date is set to the last time of the day and the next set of values are accessed from the beginning of the next day. This is done as the combined values of two days cannot be handled in the database. It creates objects for the java.util classes Calendar and SimpleDateFormat. Hence it calls a method parse_time() to parse the time in the required format.
- *parse_time():* This method parses the given date and time into integers and passes them to get_endtime to be presented in an appropriate manner.
- *SliderDemo(int):* This method acts as the parameterized constructor to create a thread to run in the background in order to get updated information from the database at regular intervals. It initializes all the variables needed for the thread to run.
- *run():* This method is used to override the run() method in the Runnable interface. This method defines the action the thread that has already been created, needs to implement. It

calls the update_array() method in order to retrieve values from the database and thus updates the array from which the values are being displayed, regularly.

- *stateChanged():* This method listens to the slider bar every time its position is changed and updates the information that is being displayed in the window depending on the position of the pointer on the slider. Since the position of the pointer on the slider is associated with an Action Listener, the actionPerformed event is called every time a change in the position of the pointer is detected.
- *startAnimation():* This method is used to start the movement of the pointer on the slider if it has stopped for any reason.
- *stopAnimation():* This method is used to stop the motion of the pointer on the slider bar.
- *actionPerformed():* This is the method which actually handles the position of the slider bar at regular intervals or when called. It also handles the action to be performed when the buttons in the display window, corresponding to the patients are clicked. For every button click event, a new window is created with personalized configuration to display the respective patient's position as a continuous stream of images. It calls a method called updatePicture() in order to update the image being displayed in the window.
- *updatePicture():* This method updates the windows being viewed, to display the image for the current frame. It creates an *imageIcon* using createImageIcon() method if it already doesn't exist. If the image already exists, it sets the corresponding image to the current image frame in order to update the frame.
- *createImageIcon():* This method creates an image icon based on the given path and returns the image URL.
- *createAndShowGUI():* This method is used to create and set up the window, add frames to the window and display the window and start the animation.
- *SliderDemo():* This method calls the above mentioned functions in order to systematically process the data and display it in the required manner. It creates the Jslider to represent the progress in time for the desired activity after the initializations to the program. It calls update_array() to update the array initially before displaying the images from it. It then creates a thread to run in the background to update the values in the array that displays the images. Once the Jslider has been initialized, the other components are added on to it to be initialized. A respective button is created for each patient and is

associated with a window listener so as to disable the button when the window is active and running. On closing the window, the button is enabled again. It also initializes the map to be displayed. Once the Jslider is initiated, it moves at regular intervals and this movement is associated with an action event that updates all the values being displayed at every second or the length of the interval.

Dbconn – This is used to retrieve values from the database in a divide and rule manner. It contains various functions for accessing various different sensor values. The methods in this class can be briefed as follows:

- *get_patients():* This method returns the patient ids of the different patients that are actively being treated.
- *get_patient_activity():* Accepting as input the start time, end time and a date for the required time interval, this method accesses the data from the database for retrieving the patient positions and location co-ordinates. Since there are multiple patients and hence multiple tables from which data is to be accessed, this function employs a while loop to iterate and execute the query. This reduces overhead on a single select statement and thus decreases the response time. It then gets the associated filename related to the body posture and returns it.
- *get_patient_data():* Similar to the above method, this takes start and end times and date as the input in addition to the length of the interval. The values for the temperature and light sensors are accessed from the database. However, the intervals at which these values are recorded do not necessarily coincide with the interval at which patient positions are recorded. Hence, these values are to be assigned a nearest value. This function processes all these values and returns values the same number of values as the above function.
- *get_filename():* This method returns the appropriate image file for the given data.
- *timeDiff():* Based on the start and end times, this method calculates the difference between them. It creates objects for the SimpleDateFormat() class and returns the difference by a measure of milliseconds which are converted into seconds before being submitted.

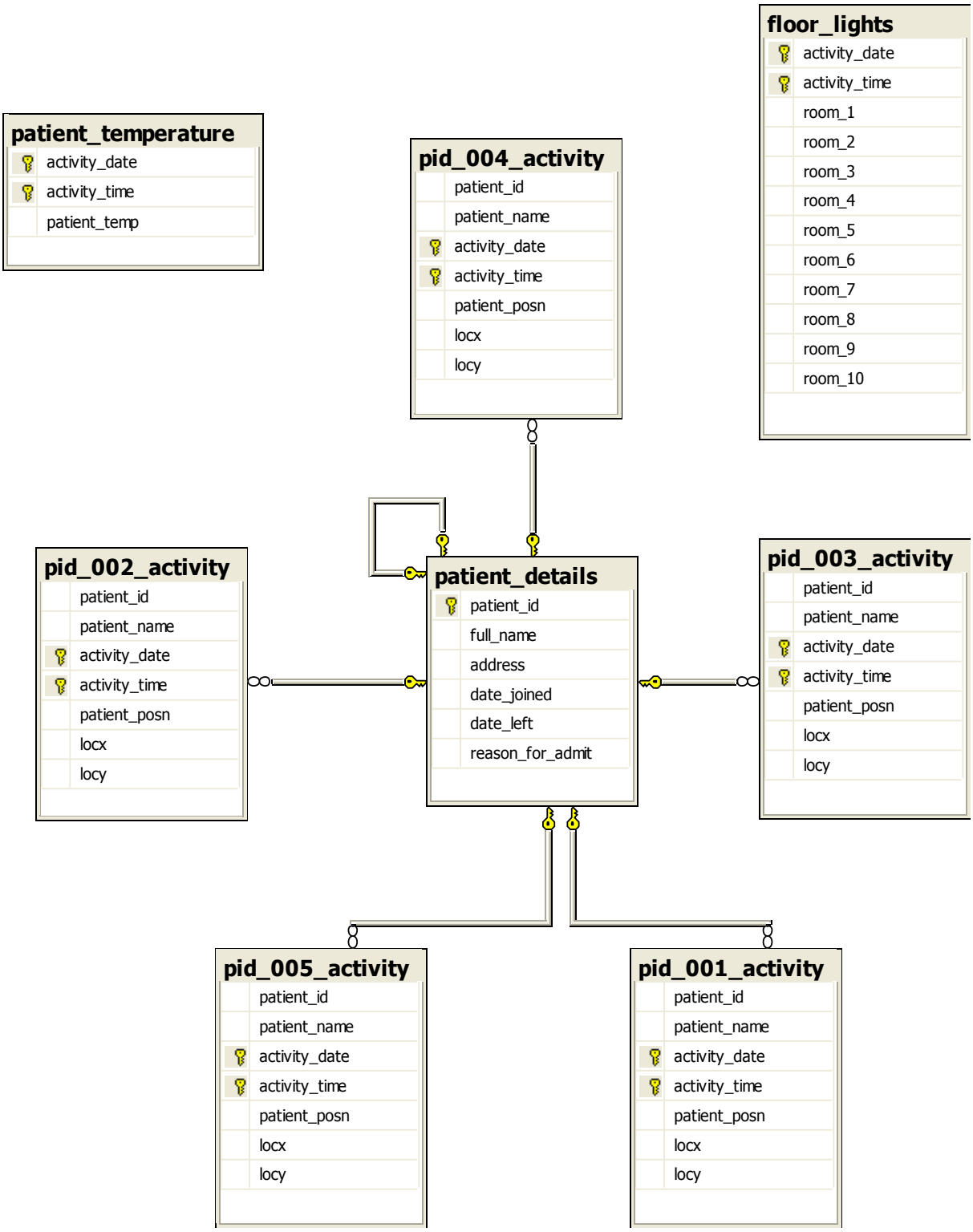
ShowMap – ShowMap is only accessed to paint the window with a map and the patient's location coordinates with appropriate ids and the status of lights onto it. It only contains one method which overrides the original method in the super class.

- *paint():* This method creates a map and represents every patient as an icon with the patient's id above the icon. It also develops a graphical display of lights in the room.

3.3 Database Diagrams

The database is the major input to this application. However, the given database must conform to a certain pattern. That pattern can be best depicted and reciprocated by a design. Thus, the following diagram that corresponds to the database diagram, represents the different schemas, the columns in each of the schemas, the relation between the schemas and also the dependencies between them.

Figure 3.3 Database Diagram


floor_lights
activity_date
activity_time
room_1
room_2
room_3
room_4
room_5
room_6
room_7
room_8
room_9
room_10
patient_temperature
activity_date
activity_time
patient_temp
pid_004_activity
patient_id
patient_name
activity_date
activity_time
patient_posn
locx
locy
pid_002_activity
patient_id
patient_name
activity_date
activity_time
patient_posn
locx
locy
patient_details
patient_id
full_name
address
date_joined
date_left
reason_for_admit
pid_003_activity
patient_id
patient_name
activity_date
activity_time
patient_posn
locx
locy
pid_005_activity
patient_id
patient_name
activity_date
activity_time
patient_posn
locx
locy
pid_001_activity
patient_id
patient_name
activity_date
activity_time
patient_posn
locx
locy

The tables displayed in the figure could be elaborated as follows:

- *patient_details*: This table is used to store the full details of all the existing/active patients in the Health Care System.
- With the patient's id from the primary key of the *patient_details* table, the other tables are created as pid_001_activity, pid_002_activity, pid_003_activity, pid_004_activity, pid_005_activity. The values pid_001, pid_002, pid_003, pid_004, pid_005 refer to the *patient_id* in the *patient_details* table.
- *floor_lights*: This table is used to record the status of lights in each room through the light sensors at regular intervals.
- *patient_temperature*: This table records the room temperature at regular intervals.

CHAPTER 4 - Implementation

The suggested design architecture is translated into code during the implementation process. The implementation process converts the underlying architecture into logical code in order to produce the desired results. This chapter explains how the logic behind the application is implemented beginning from accepting an input to displaying the output. This chapter also includes some screen shots for a better visualization purpose.

4.1 Input

The application has two major inputs. One is the user input which specifies the time bounds between which the activity is to be time lined and the other is the input to the application. They can be explained in detail as follows:

4.1.1 Input to application

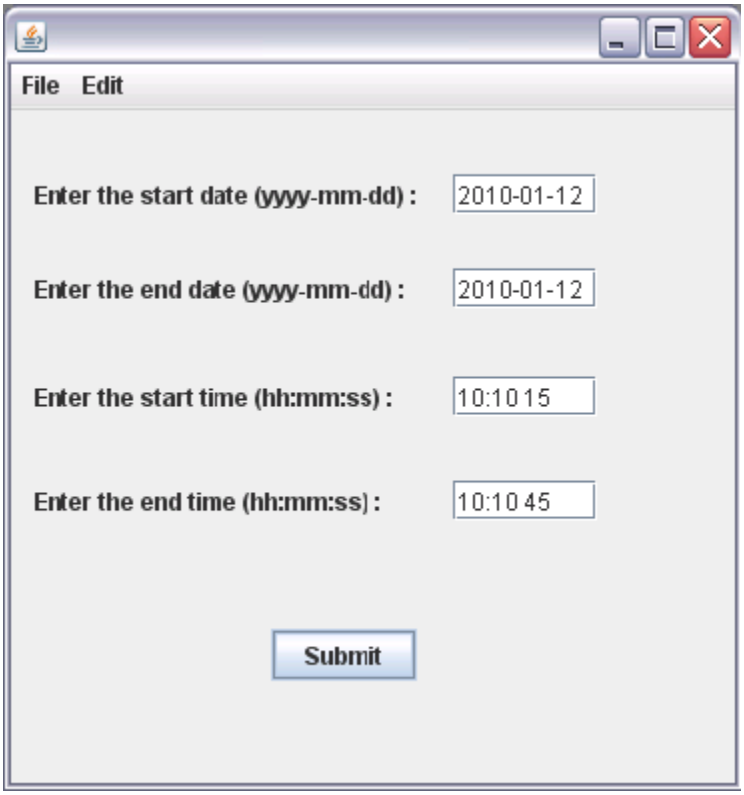
The application that handles recording the patient's movements by equipping the patient with sensors records the values and displays the patient positions, simultaneously storing them in the database. Since the current application is an extension to it, the current system takes that application's output as its input. Hence the application assumes the image files related to the patient's different positions to be known in advance. Apart from this, the image of a blue print model of the Health Care System is to be given to the application. In addition to the image files, the values corresponding to all the different sensors at every instance of time are to be given. In other words, the application needs a database to access the data from. Thus the database accords to the major input to this application.

4.1.2 User Input

Having received the necessary inputs, the application now requires only the time frame to be specified in order to start processing. The system hence takes the time constraints which correspond to the start date of the activity, end date of the activity, the time on the start date from which it is to be visualized and the time till on the end date till which the activity is to be visualized. Apart from these inputs, the application can also be configured to take as input the

rate at which the slider moves, the different sensor values to be displayed etc. The following is a screenshot of the input display window that accepts the user inputs.

Figure 4.1 User Input - Screen shot



A set of input values have also been entered for a better visualization.

4.2 Deployment

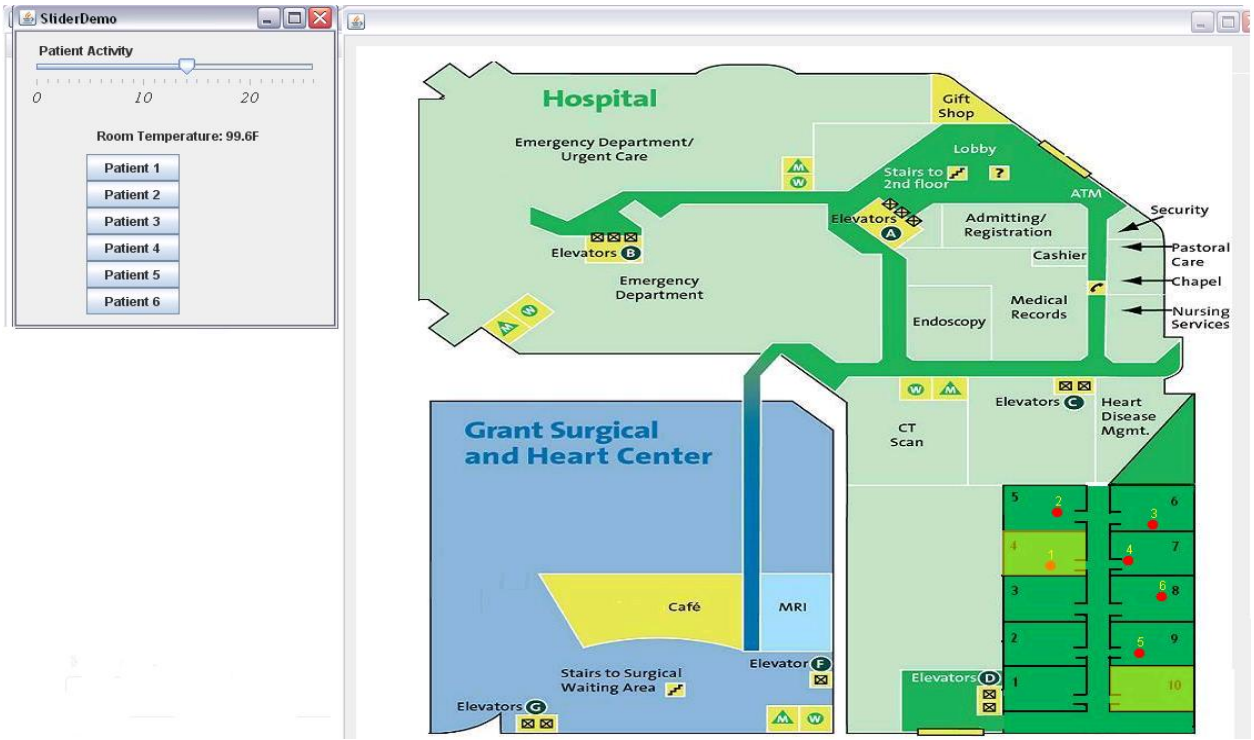
The deployment gives the complete details of the execution of the project. The functions of each of the methods in the application and the execution details can be listed as follows:

Once the inputs are accepted, the start and end dates are validated to check if they are in the desired format. Similarly, the start and end times are validated for the required formats. If valid, it is also checked if the start date and time are prior to the end date and times to get legitimate values from the database. After validations, the values are sent to SliderDemo class for further processing.

In the Slider Demo class, based on the time difference between the start and end times, an interval is determined to split the values to be sent to the database. Once this is done, the start and end times for that interval are calculated. If the end date exceeds the existing date, it is truncated till the end of the day for ease of use in retrieving the values from the database. The patient positions, location coordinates, temperature, status of lights are all retrieved from the database and stored in separate arrays. The temperature and status of light values might not be recorded at the same interval as the patient positions and coordinates are being recorded. Hence after retrieval of the values from the database, the missing values are adjusted to the nearest possible values. Thus all the arrays when updated are synchronized.

Since the array is now updated and contains some values, the slider is initialized and started. In the process of initialization, it creates a separate window and calls the ShowMap class to display the map in the new window with the updated values of patients, corresponding ids, their location and the status of lights. The temperature is displayed on the slider itself. The following is a screen shot of the application running with a few patients located and shown on the map with the status of lights and corresponding temperature value at that instance.

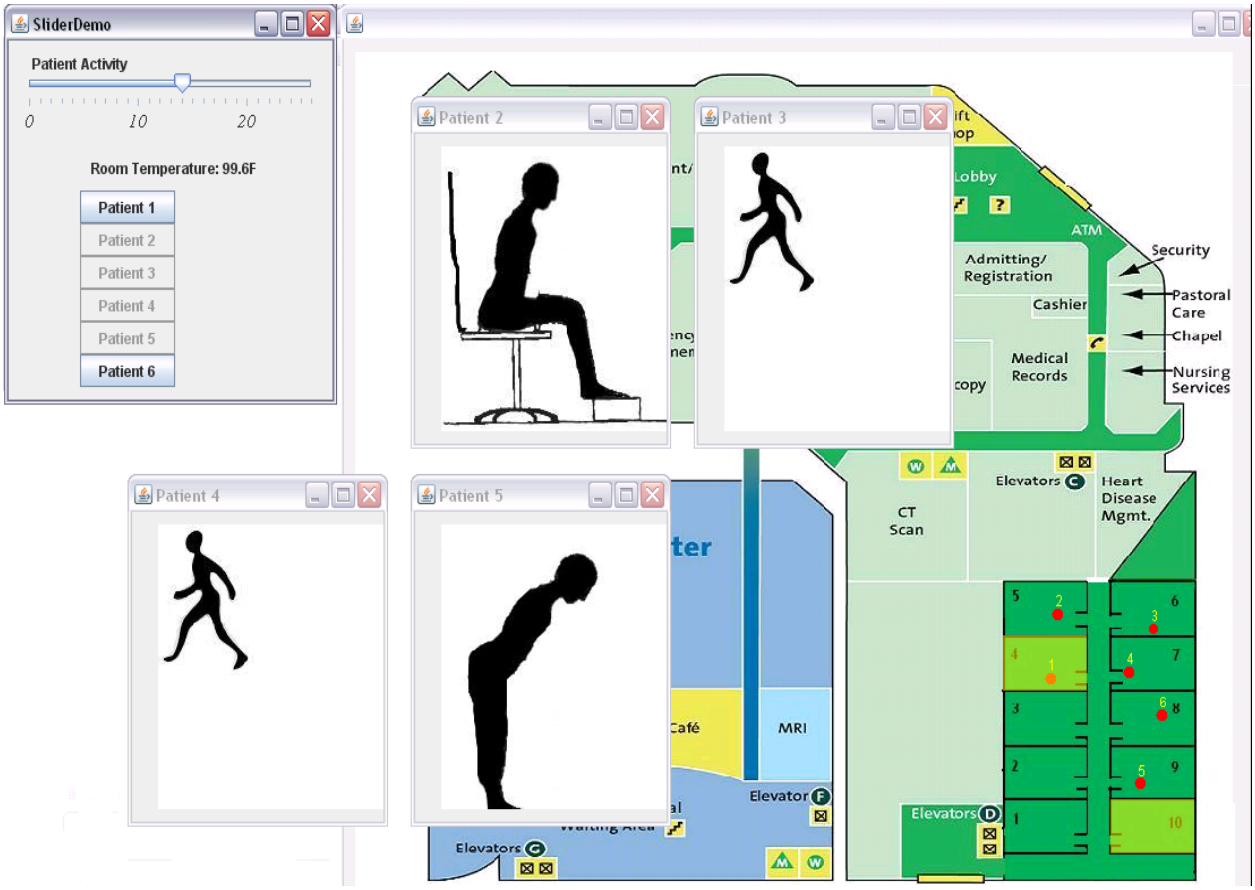
Figure 4.2 Screen shot of the application in a running state with a few initial values



The above diagram represents a basic model of the application, running with six patients being representing by the red icons with an associated patient id above them. The fluorescent color filled inside the rooms indicates that the lights in room 4 and 10 are switched ON. The Slider is in progress while the temperature is displayed beneath it.

As the pointer on the slider progresses with time, the values of the entities change and have to be updated. Hence the action event associated with the slider is called every time its position changes and the entities being displayed like the images, location coordinates, status of lights etc., are updated from their respective arrays. The action event is also associated with a button along with a window listener for each patient. On clicking the button, the respective patient's position can be viewed. All these values are always in synchronization with the slider. The following is a screen shot of the application in a completely running view of all its entities.

Figure 4.3 Screen shot of the application in a running state with all entities



The above diagram represents a running state of the application with six patients being shown on the map along with the positions of selected patients 2, 3, 4, 5 being displayed in individual windows. It can be noticed that the different patients have different positions and also that the buttons representing the patients being displayed are disabled. This has been implemented in order to avoid redundant windows for a particular patient's activity. On closing a particular window, the respective patient's button is again enabled in order to be able to view the patient's activity any number of times. In the actual deployment, it can be observed that the patient's movement is in synchronization with the icon of the patient depicted on a map.

A thread runs in the background in order to update the values of the array so that a null value is not displayed in the output. The thread makes periodic calls to the database until the array has been updated completely.

A patient's position might not be monitored in a continuous manner. A user can choose to have customized view only for specific instances of time. Hence a drag feature has been implemented on the slider, in order to not represent a null value if the thread has not updated the array with the values that correspond to the position of the slider when dragged. When the action performed event is fired due to a change in the position of the slider when dragged, the application first checks if the array values corresponding to that position are null. If null, another thread is allocated to retrieve values from the database for that instance of time. Hence the array is now updated and contains values to be displayed. The application now has two threads running for updating the array at a quicker pace.

The output is thus, always synchronized with the slider and the values are displayed promptly.

CHAPTER 5 - Testing

An application's performance and functionality can only be determined and proved by testing it with discrete and numerous values. Only through intensive testing, can the capacity of the amount of data an application can handle, be determined. Thus, this chapter aims at describing the amount of load, the application can handle. For ease of testing, the application is tested in different ways. The following sections list them.

5.1 Unit Testing

Unit testing plays an important role in handling all the minor details in a project's functionality. It is responsible for testing every test case and every path the code could take. This section currently aims at testing the different possible test cases in every module of the project. Unit testing has been performed during development of the application and the relevant loopholes have been handled accordingly. The following section indicates some of the cases that were handled.

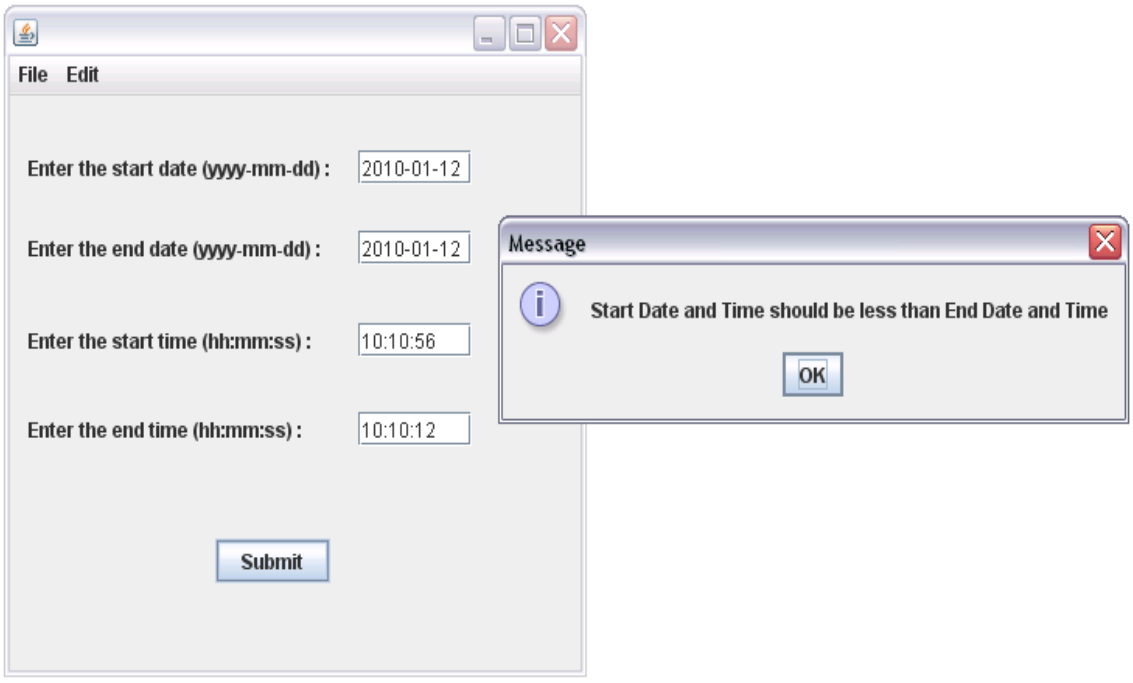
- The start and end dates are validated for the desired format. Any format other than the required format causes an alert box to be populated to inform the user about the required format.

- The start and end times are also validated for the desired formats. Similar to the above case, an alert window is populated asking the user to check the format of the time entered.

- Apart from the date and time validations, it should also be checked if the start date and time are less than the end date and time in order to provide a legitimate input to the database. Thus an alert window is once again popped up if this rule is violated asking the user to enter a start time less than the end time.

The following figure represents a screenshot of the testing of this module.

Figure 5.1 Screen shot of a unit test case



- In addition to the input values, the get_data() method that retrieves the temperature and status of light values has also been well tested by providing a different interval for every test case. Thus, cases with pre-adjustments and post adjustments to the array in the case of the start value of the particular entity not coinciding with the start of the interval, have also been handled, including the case with no adjustments.

- After splitting the time into intervals, the end time for every interval is to be calculated carefully, simultaneously comparing it with the end time of the day, end time of the activity etc,. These cases have been handled with every possible input for the dates and the times to check the code thoroughly.

- The functionality of the drag feature associated with the slider could be one of the major components of Unit testing. For this purpose, the action listener for the slider has been well tested by giving in large input values and thus forcing multiple threads to run in the background in order to get the array updated without any latency when it is dragged.

5.2 Integration Testing

Integration testing accounts to the testing performed after combining all the individual modules in the application. It is conducted after performing unit testing on all the individual modules in the application and then combining them. After the unit testing was performed on the

above mentioned modules, all the features were integrated and were tested collectively to ensure the performance of the system. Certain unexpected results generated on the integration of the modules have been handled.

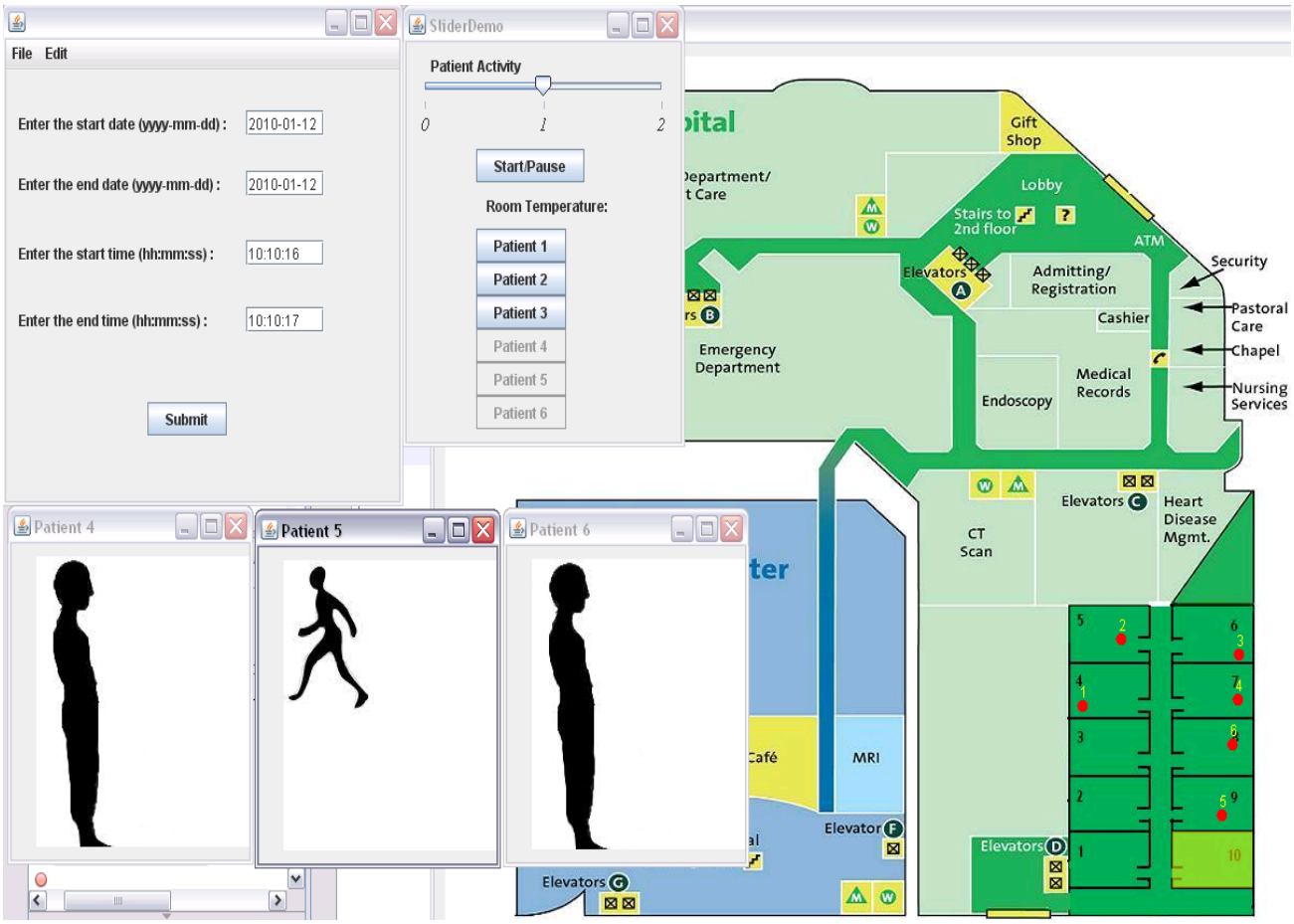
5.3 Correctness Testing

After having tested the entire application as an integrated entity, the system has been subjected to correctness testing in order to determine the proper functionality of certain features. This section includes testing the action listeners, the drag feature which involves creating and initiating a thread every time the array has not been updated at the corresponding frame number when the slider has been dragged to a certain position much ahead of time, testing the display function which rounds off the values in the temperature and status of lights table in order to display them at every second and testing the correct functionality of the system with a very high load. The following test cases establish the correctness of each of the features discussed above.

5.3.1 Test Case 1

Starting to test the system's performance with the lowest of all values it can handle, an input time frame of length one sec has been given to the system since any value less than that would not be considered as a valid input by the system. However, the start time and end time account to two values and since the application considers both, the minimum interval would be two. The temperature and status of light values have been recorded at certain intervals which were certainly greater than one. Hence an input time frame has been chosen in such a way that there exist no values for that instant of time in the temperature table. Thus the output consists of portraying the blueprint model of the Health Care System with the display of the status of lights and no temperature. The following is a screen shot of it.

Figure 5.2 Screenshot of correctness testing with a minimum load



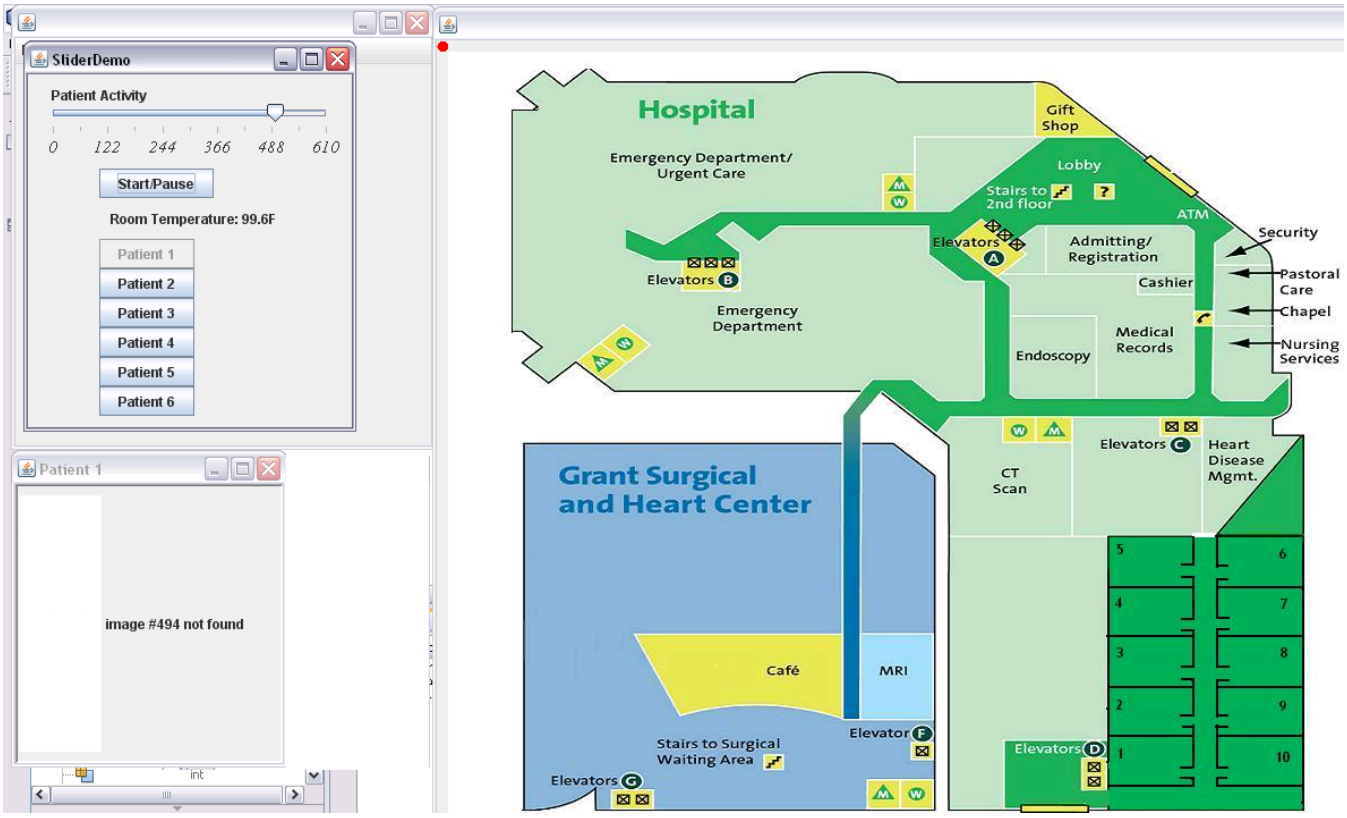
The null value for temperature indicates that no value of temperature has been recorded for that instant of time. Since there are no more values retrieved from the database for the temperature to be rounded off to the nearest value, no value is represented. The status of lights has been recorded for every two seconds. Hence the screenshot shows the display of lights rounded off to the nearest possible values. Thus this proves that the application definitely displays the relevant values for lights even when there is no value for temperature.

5.3.2 Test Case 2

Testing the system with a considerably low load of few minutes leads to the conclusion that the system definitely becomes unable to handle this amount of data without any latency when forwarding the slider to a large amount of time on disabling the drag feature. To prove this, the system has been tested by disabling the functionality of the thread that immediately retrieves the values from the database when the slider has been dragged to a considerably forwarded

position. The following screenshot proves that the thread created during the initial phase of the application is not sufficient to handle the complete set of values from the database before the actual execution of the application starts.

Figure 5.3 Screenshot of correctness testing of a certain feature



The message indicated in the patient's position window indicates that the array hasn't been updated in time to display the position of the patient. The same is the case with the display of lights and the patient's coordinates. The latency in viewing the desired display of results in such a case is very high. This threshold depends on the configuration of the system on which the application is running as well as the interval based on which the values are retrieved from the database. Hence, to overcome this deficiency and to display the positions of the patients without any latency when the slider is dragged forward in time for a considerably large time frame, another thread is created to retrieve the values from the database in short intervals. Testing the

system with a comparatively heavy load determines the performance of this particular feature of the system.

5.3.4 Test Case 3

The system is bound to reach a saturation point after which it cannot handle an increase in the amount of load given to it. However, the application can definitely handle values close to two days. The following screenshot indicates its performance for a load value close to a single day. The input indicates that the start and end dates are different, unlike the previous cases.

Figure 5.4 Screenshot of correctness testing under heavy load

The system still manages to perform efficiently under this amount of load. Hence based on observation, the maximum load the system can take has been approximated to two days taking into account the capacity of the array that holds these values.

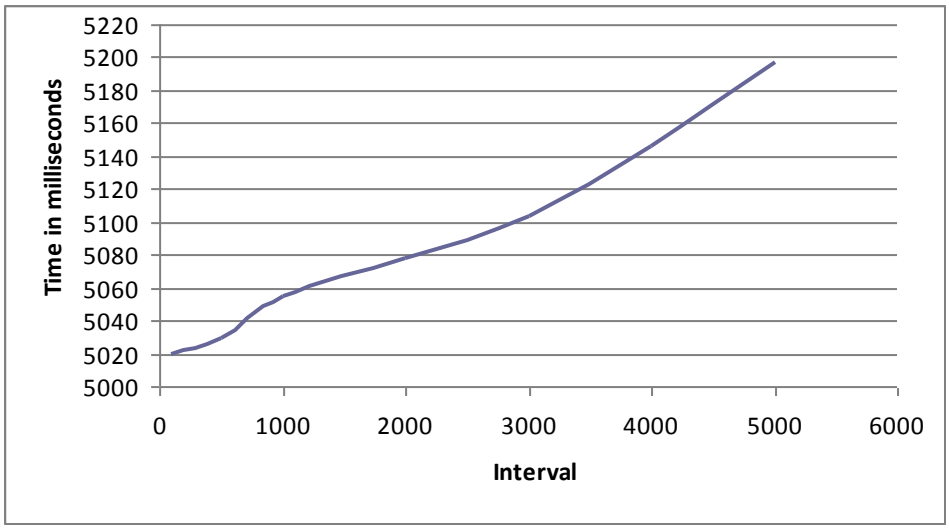
5.4 Performance Testing

The application is subjected to intensive testing during performance testing. The system's tolerance is put to test under excessive load and its performance under these conditions, is evaluated in order to determine the system's optimal performance and the response time for different loads. The test cases in the following sections have been chosen such that they best describe the system's performance under exceptional cases. The system's performance has been establishing by varying the different factors that could have an impact on its performance. For a better understanding of the performance of the system, graphs have been plotted based on the response time on varying different entities.

5.4.1 Test Case 1

This test case determines the time taken to retrieve the values from the database for different intervals of time. The interval corresponds to the number of rows retrieved from the database at one instant of time. The system was given a load of up to five hours which constitutes to 18000 seconds which in turn corresponds to retrieval of 18000 rows from the database, eventually in shorter intervals. In the running mode, the application randomly determines the number of rows corresponding to one interval and this interval is constant for a given duration. However, in the following test case, this interval has been changed considerably and the time taken to retrieve the corresponding number of rows from the database has been determined for six patients. The number of rows retrieved from the database is shown on the x-axis and the time (in milliseconds) taken to retrieve the corresponding number of rows has been shown on y-axis. It yielded a considerably non-linear graph which can be represented as follows.

Figure 5.5 Graph plotted for number of rows retrieved against time taken in milliseconds

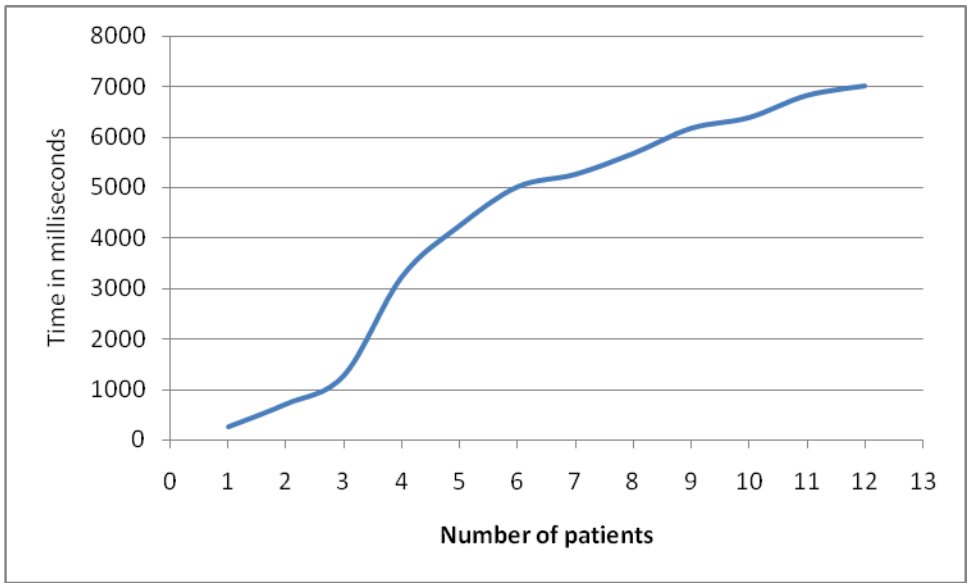


The values on the graph have been plotted by taking an average of around 100 values for each interval. It can be observed that the time taken to retrieve does not change considerably from 100 to 500 rows since there is not much difference in time in retrieving 100 rows and 500 rows for a very large database. However, the time taken certainly increases when compared to 2000 and 3000 rows. Though the retrieval of rows is done in a serial manner for every patient in the database, the length of the interval does not seem to have a substantial effect on the output as the difference between the highest and lowest times is only 200 milliseconds approximately.

5.4.2 Test Case 2

This test case determines the time taken to retrieve values from the database for different number of patients in the database. Each patient has a separate table in which the patient's activity is recorded. Hence increase in the number of patients constitutes to the increase in the number of tables from which the values are retrieved and hence more response time. Thus when these values are plotted on a graph, a gradually increasing curve can be expected. This test case represents the behavior of the system when the number of patients is increased gradually for four hours duration of the activity. The following graph plots the number of patients on the x-axis and the time taken (in milliseconds) to retrieve the values from the database for the corresponding number of patients on the y-axis.

Figure 5.6 Graph plotted for time taken to retrieve values against number of patients

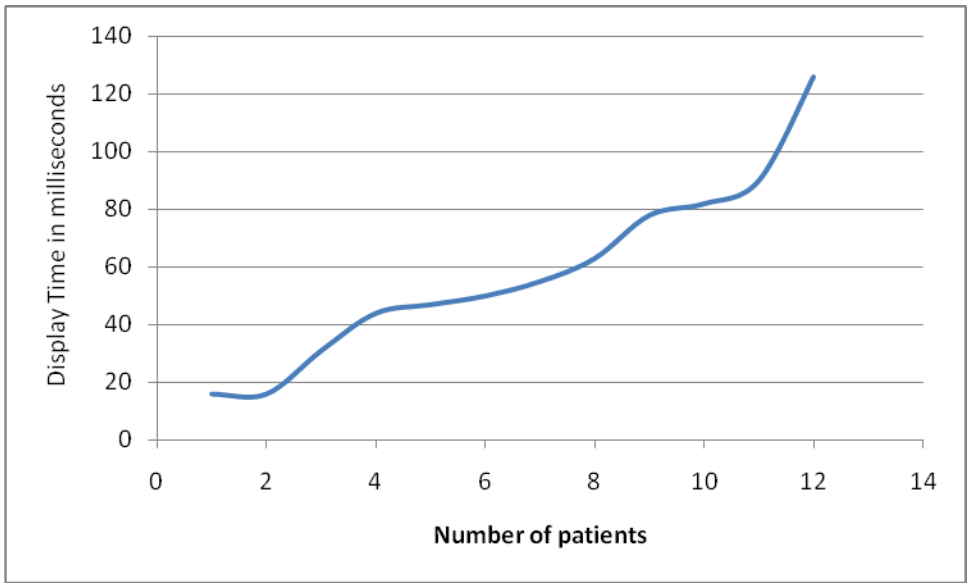


As expected the graph has been non-linear. The time shown in the graph is the time taken to retrieve values from the database for a certain interval. The interval in this case is 100 which corresponds to the number of rows retrieved during each retrieval. The time represented in the graph for each patient is an average of 100 such values for that interval. The values are retrieved from the database in a serial order. Hence for every additional patient, the time increases by a considerate amount and adds to total time. This is the factor that constitutes to the increase in the graph in a non-linear manner.

5.4.3 Test Case 3

This test case determines the time taken to display the values on the screen for different number of patients. Every patient in the system is attributed with a separate window with window listeners associated with it to view the patient's activity. Hence though this is done during initialization, every patient's window is to be updated at every second during the execution of the application. Hence more number of patients in the system constitutes to more time in updating the frame for every patient and thus constitutes to a greater display time or the time taken to display the values. Though it is negligible, there definitely is a gradual increase in the display time with more number of patients. Hence the same has been plotted as a graph with the number of patients on the x-axis and the display time in milliseconds on the y-axis.

Figure 5.7 Graph plotted for display time against the number of patients

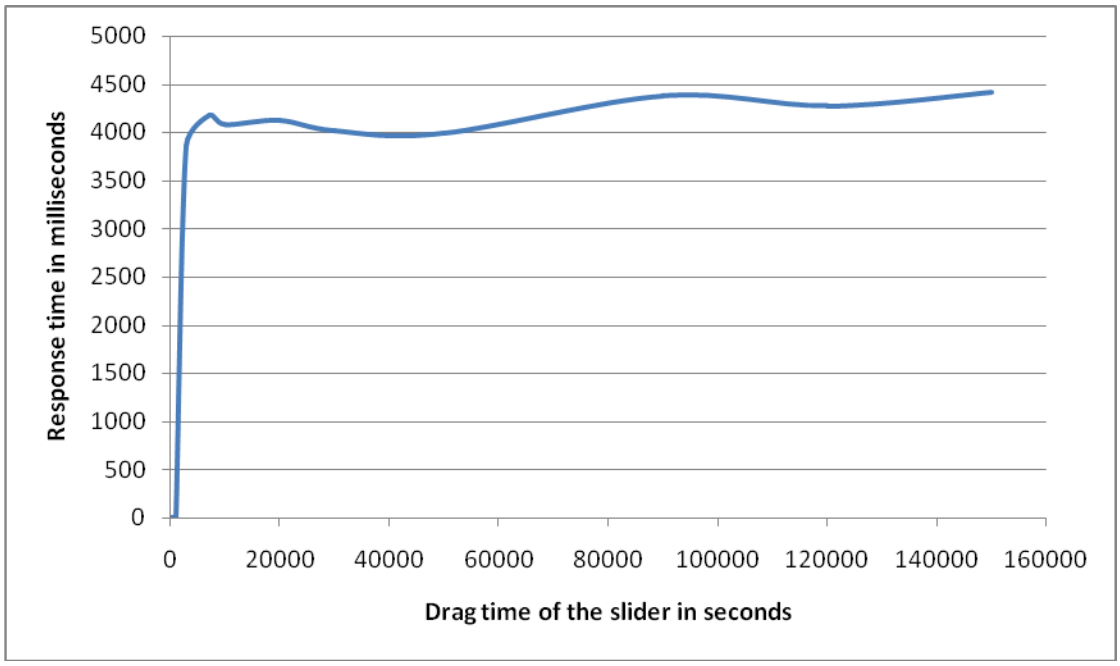


As it can be observed from the graph, the display time increases linearly with increase in the number of patients. The time in the graph corresponds to the time taken in milliseconds. During the execution of the application, for every second or for every change in the slider's position, the frame representing the patient's activity has to be updated for every patient along with the patient's location on the map. Since this accounts to a certain overhead on time and since the increase in display time is directly proportional to the number of patients, a linearly increasing graph has been observed.

5.4.4 Test Case 4

This test case determines the time taken to retrieve and display the values when the slider is dragged ahead of time by a certain amount. The system was given a load of around two days for five patients and with a retrieval rate of 300 rows at an instant and the response time was calculated when sliding the slider ahead of time. The number of seconds by which the slider has been dragged has been varied and the response time for those values has been calculated. This test case determines the impact of the amount of drag on the system. A graph has been plotted with the above mentioned variation in the amount of drag which can be represented as follows. The following graph represents the time (in seconds) by which the slider is dragged on x-axis and the response time (in milliseconds) to retrieve and display the values on the y-axis.

Figure 5.8 Graph plotted for drag time of the slider against response time



As observed, the graph is not a smooth curve as and increases suddenly for a drag time of 2500 seconds. A thread runs in the background to update an array from which the values are displayed. Hence the array needs to be updated for a value to be displayed. However, if the slider is dragged to a position to a lengthy amount ahead of its current position for which the array is not updated, another thread is created to update the array from that position so that there is almost negligible response time before the values are displayed. However, even for this thread to retrieve the values and display them, takes a certain amount of time. Hence in the above graph, there is a zero response time up to 2500 seconds of drag and the response time increases to 4000 milliseconds approximately after that. The threshold in this case is 2500 seconds as this value depends on the interval or the number of rows being retrieved from the database. As in this case 300 rows are being retrieved from the database, 300 seconds are updated at one instant. Hence the first thread can update to approximately 2500 seconds by the time the application starts and the slider is dragged. However, since in this case, a second thread is to be created to retrieve the values from the database after 2500 seconds, the response time after 2500 seconds is approximately 4000 milliseconds. This response time does not vary much for any amount of drag after 2500 seconds as it is always the time taken to retrieve 300 rows from that position of the

slider. The threshold certainly varies on the interval. However, the graph always takes a similar shape irrespective of the length of the interval.

CHAPTER 6 - Conclusions & Future Work

6.1 Conclusion

The project was aimed at developing a functionality that allowed visualization of a patient's activity in accordance with the surrounding environment. This goal has been accomplished as a graphical user interface has been developed, that allows a customized visualization of the patient's positions in separate windows along with the patient's location displayed on a map of the building, the room temperature values, the status of lights in the room, all in a synchronized mode with a slider, thus producing continuous streaming of the patient's activity.

6.2 Limitations

The application has been developed with a consistency and flexibility that assures ease of use and optimal performance with minimal effort. Its comprehensive monitoring capabilities provide unparalleled flexibility in monitoring the patient's needs at any point of time. However, the system meets certain limitations in its performance. The system cannot be configured to these limitations with any number of improvements in the underlying architecture. These can be presented in details as follows:

- The system cannot handle real-time values as there needs to be data in the database which is the input to the application. However, the system can get very close to displaying real-time data values.
- The system is expected to handle data values, recorded each second, continuously for two days and not more. The arrays that store the entities to be displayed have a maximum limit on their size and do not allow for a higher storage.

6.3 Future Work

The application can further be enhanced on its functionalities to overcome some of the limitations mentioned above. Its well built architecture allows the addition of several new components as well as modification of the existing components without much hassle. The project can be further improved by appending some of the following functionalities:

- Additional functionalities can be added in order to display more number of entities associated with the patient or the surroundings like the blood pressure, body temperature, etc.
- The application can be collaborated with another application so as to produce a resultant system that can monitor the patient's activity as well as store it for future access.
- Multiple arrays can be used to handle the display of data for every entity, in a synchronized manner, in order to be able to increase the application's capacity of handling load.

References

[1] J2SE

http://en.wikipedia.org/wiki/Java_Platform,_Standard_Edition

[2] Swing

http://en.wikipedia.org/wiki/Swing_%28Java%29

[3] Jslider

http://v1.dione.zcu.cz/java/docs/swing-1.0.3/doc/api/com/sun/java/swing/JSlider.html

[4] Java FX

http://java.sun.com/javafx/

[5] JDBC

http://java.sun.com/docs/books/tutorial/jdbc/overview/index.html\

[6] MySQL

http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html

[7] Architecture

http://java.sun.com/products/jfc/tsc/articles/architecture

[8] NetBeans – Documentation and Tutorials

http://netbeans.org/kb/index.html

[9] Swing Utilities

http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/SwingUtilities.html

[10] Date and Time Validations

http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html

[11] Painting in AWT and Swing

http://java.sun.com/products/jfc/tsc/articles/painting/

[12] Window Listeners

http://java.sun.com/docs/books/tutorial/uiswing/events/windowlistener.html

[13] JFrames

http://java.sun.com/docs/books/tutorial/uiswing/components/frame.html

[14] JComponent

http://java.sun.com/javase/6/docs/api/javax/swing/JComponent.html

[15] Calendar – Parsing dates

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Calendar.html

[16] A blueprint model of a Health Care System

http://www.ohiohealth.com/images/grant/maps/GrantMaps_secondfloor.jpg